Software Security Final Project

Group name: AppArmor Access

Members: Saire Conejo, Santiago Ajala and Patricio Mendoza

Part A: Implementing a Secure API

The creation of our application arises from the need to track missing persons before, during and after a natural disaster. This will help to provide prompt assistance to those affected and save many lives.

Methodology:

To start with the system we need to know the position of a person so the first step is to have a mobile application that provide the localization. For that pospuse we have to implement a Aplication Program Interface (API) that will help to work with the information of the users. This API allows users to safe their position in the database an then the administrator can work with this information.

API

It is implemented using Flask module in python. Basically this API manages and retrieving geospatial data points. This application uses an MySQL database called '*puntos.db*' to storage geospatial data and users. There are two tables ***Usuario*** and ***Punto.*** In the ***Usuario*** table there are the following columns ***id*** which refers to the ID card number, ***nombre***, and ***apellido,*** these refers to name and the last naem of the users respectively. In the table ***Punto*** there are following columns: ***id, latitud, longitud**, and **fecha.***

API Documentation

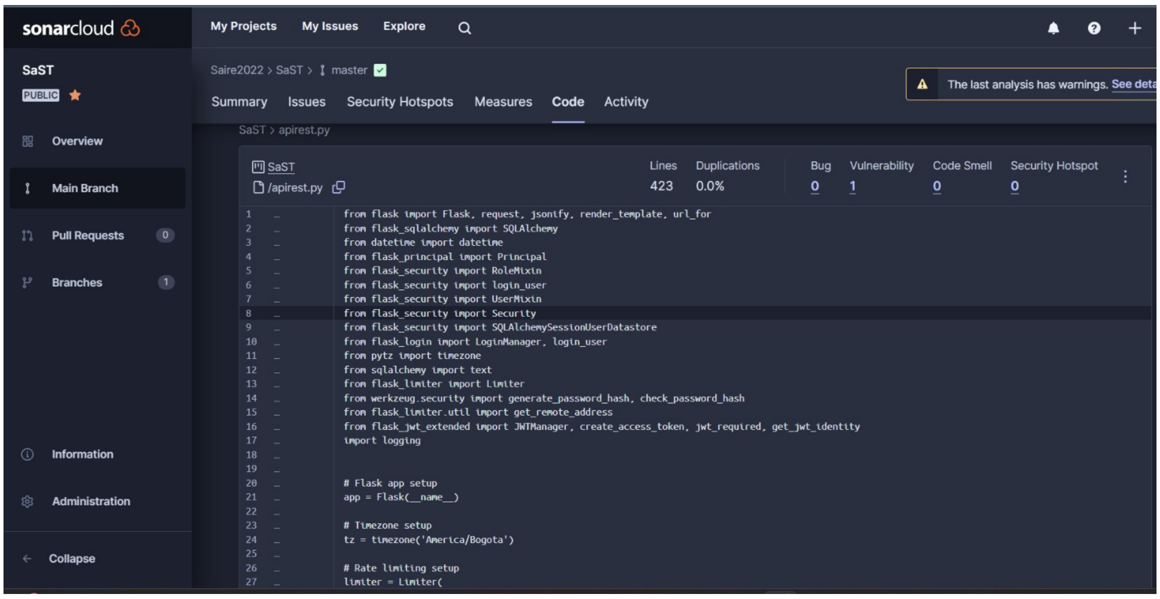
Base UL:
https://documenter.getpostman.com/view/27080186/2s9YkgEkzd?fbclid=IwAR2vG1nhEZBBcy1Js_1UptX0guTxNmz0o3VRXSwd_RjRkICoLsRPaYcjUdk

Notes

- All endpoints return data in JSON format.

To make sure that our api is fully secure and that the user information shared is not compromised, we will perform a Static Application Security Testing (SAST) which helps us to see the possible vulnerabilities.

For this purpose the Solarcloud tool was use and the following results were obtained.



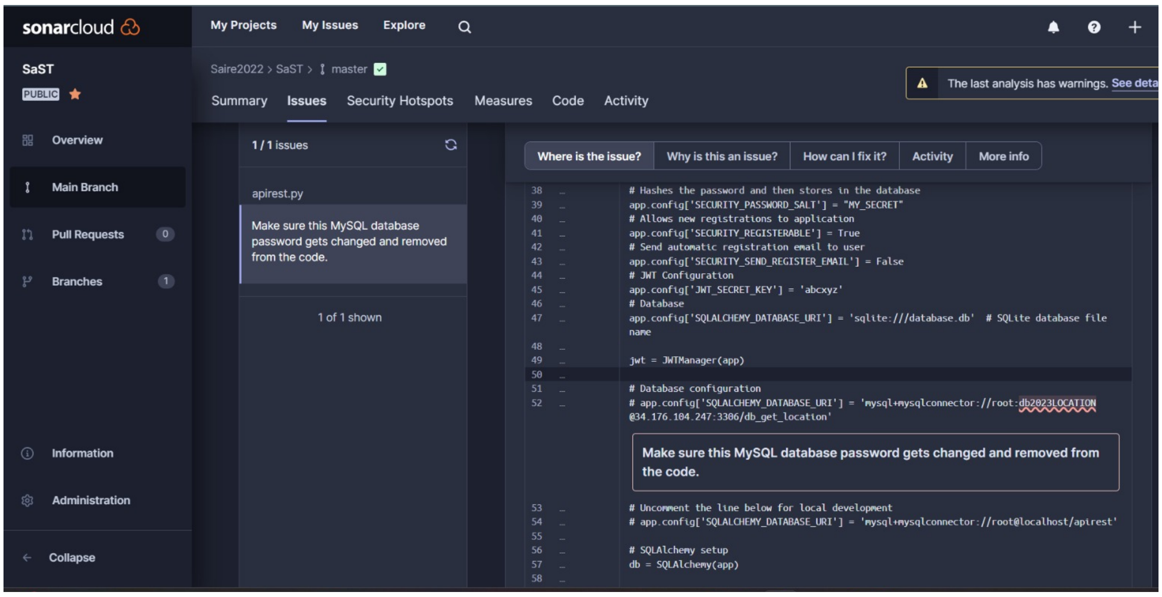
```
from flask import Flask, request, jsonify, render_template, url_for
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
from flask_principal import Principal
from flask_security import RoleMixin
from flask_security import login_user
from flask_security import UserMixin
from flask_security import Security
from flask_security import SQLAlchemySessionUserDatastore
from flask_login import LoginManager, login_user
from pytz import timezone
from sqlalchemy import text
from flask_limiter import Limiter
from werkzeug.security import generate_password_hash, check_password_hash
from flask_limiter.util import get_remote_address
from flask_jwt_extended import JWTManager, create_access_token, jwt_required, get_jwt_identity
import logging


# Flask app setup
app = Flask(__name__)

# Timezone setup
tz = timezone('America/Bogota')

# Rate limiting setup
limiter = Limiter(
```

Figure 1



Make sure this MySQL database password gets changed and removed from the code.

```
# Hashes the password and then stores in the database
app.config['SECURITY_PASSWORD_SALT'] = "MY_SECRET"
# Allows new registrations to application
app.config['SECURITY_REGISTERABLE'] = True
# Send automatic registration email to user
app.config['SECURITY_SEND_REGISTER_EMAIL'] = False
# JWT Configuration
app.config['JWT_SECRET_KEY'] = 'abcxyz'
# Database
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'  # SQLite database file
name

jwt = JWTManager(app)


# Database configuration
# app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+mysqlconnector://root:db2023LOCATION
@34.176.104.247:3306/db_get_location'
```

Make sure this MySQL database password gets changed and removed from the code.

```
# Uncomment the line below for local development
# app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+mysqlconnector://root@localhost/apirest'

# SQLAlchemy setup
db = SQLAlchemy(app)

```

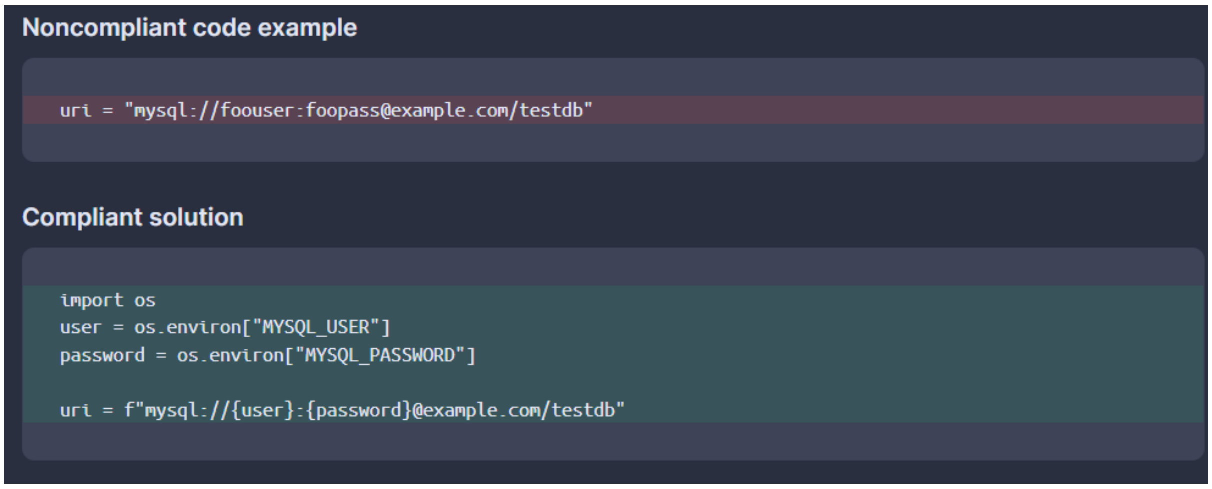
Figure 2

Sonarcloud shows us a vulnerability in the *Figure 1*. If we open this vulnerability we can see that in the *Figure 2* the next message: "*Make sure this MySQL database password gets changed and removed from the code*".

The solution that Sonarcloud suggests is:

- Hard-coding secrets in the source code, especially default values, is discouraged. This practice can lead to security vulnerabilities. Instead, the recommendation is to use environment variables or a configuration management system to handle secrets dynamically.

- The noncompliant code example shows a hard-coded URI with a username and password. The compliant solution demonstrates using environment variables to retrieve sensitive information, promoting better security practices.



Noncompliant code example

```
uri = "mysql://foouser:foopass@example.com/testdb"
```

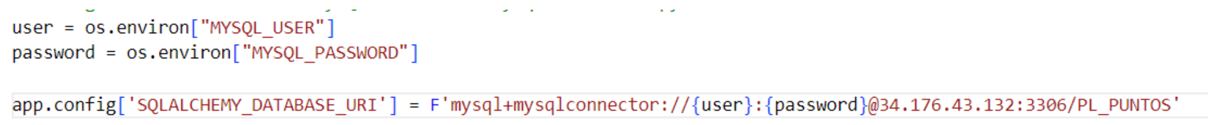
Compliant solution

```
import os
user = os.environ["MYSQL_USER"]
password = os.environ["MYSQL_PASSWORD"]

uri = f"mysql://{user}:{password}@example.com/testdb"
```

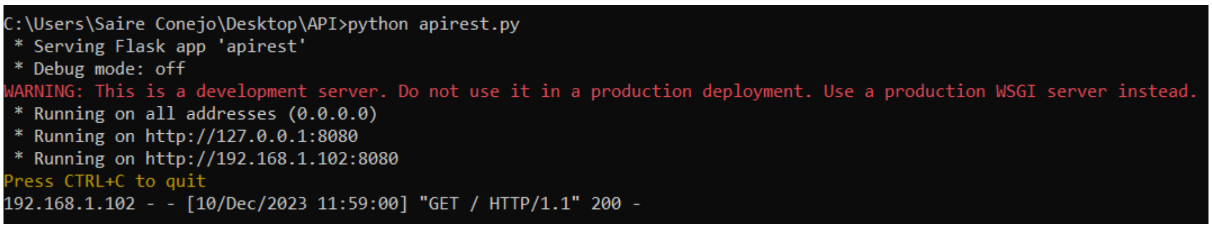
Figure 3

After set the environment variables (*Figure 4*), we have to test again and the results in the *Figure 5,* and it was successfully pass the test .



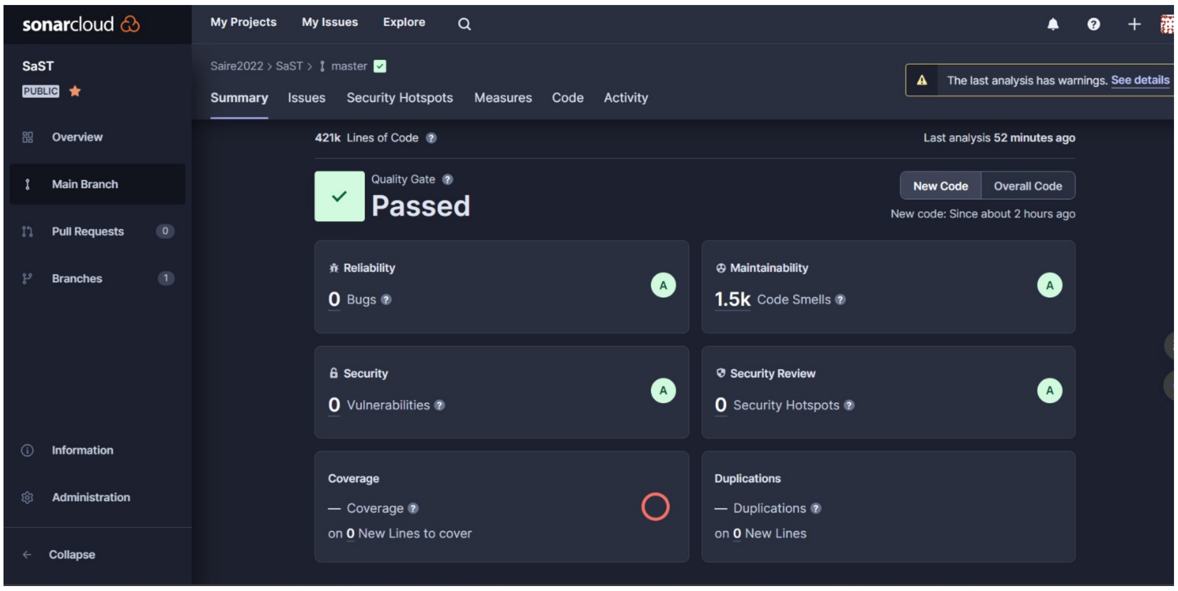
```
user = os.environ["MYSQL_USER"]
password = os.environ["MYSQL_PASSWORD"]

app.config['SQLALCHEMY_DATABASE_URI'] = F'mysql+mysqlconnector://{user}:{password}@34.176.43.132:3306/PL_PUNTOS'
```



```
C:\Users\Saire Conejo\Desktop\API>python apirest.py
 * Serving Flask app 'apirest'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://192.168.1.102:8080
Press CTRL+C to quit
192.168.1.102 - - [10/Dec/2023 11:59:00] "GET / HTTP/1.1" 200 -
```

Figure 4



The last analysis has warnings. See details

421k Lines of Code

Last analysis 52 minutes ago

Quality Gate Passed

New code: Since about 2 hours ago

Reliability: 0 Bugs — A

Maintainability: 1.5k Code Smells — A

Security: 0 Vulnerabilities — A

Security Review: 0 Security Hotspots — A

Coverage: — Coverage on 0 New Lines to cover

Duplications: — Duplications on 0 New Lines

Figure 5

Part B: Secure App

For this we will use the security API implemented in Part A in a mobile application.

Mobile application

The main objective of the mobile application is to track people by obtaining both their longitude and latitude. To implement the application, we use Android Studio and Kotlin. The application interface contains 4 screens. These screens help us verify the correct functioning of our API by incorporating the corresponding layers to make it a secure API.

The first screen (Figure 1) assists in entering the application, containing two fields required for entry. In this section, we implement the authentication layer. The user can enter only if their credentials are correct. Additionally, this part includes the access control layer; if the entering user is an administrator, they will be directed to the screen for obtaining points (Figure 2). This screen has a field where the user ID of the user from whom we want to obtain points must be entered and also a button which allows you to exit and directs you to the login screen.

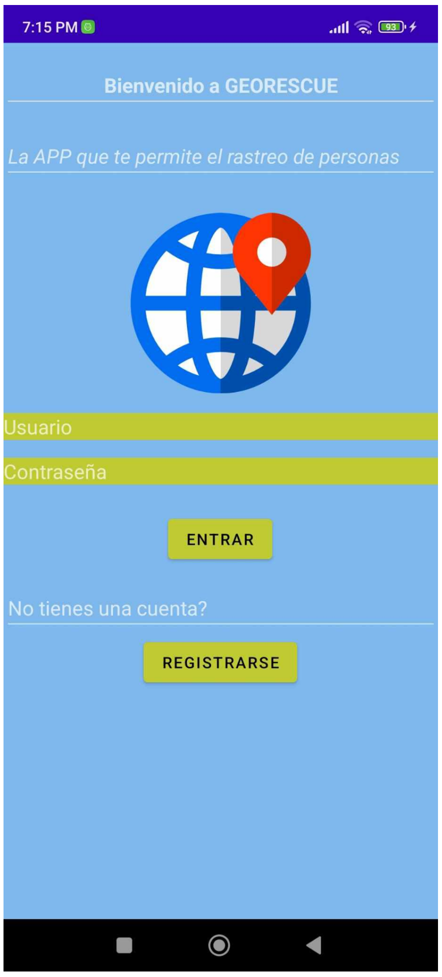


Figure 1. Login Screen.

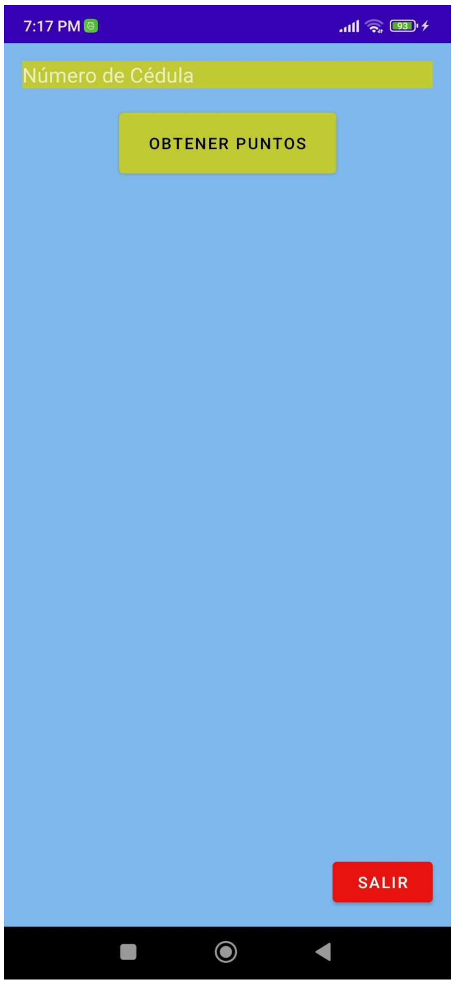


Figure 2. Get Points Screen.

On the other hand, if the user is a regular user, they will be directed to the screen for obtaining location (Figure 3). This screen has a text space displaying both the longitude and latitude of the user, along with four buttons. The first button helps us get the user's location, initiating an automatic retrieval that updates after a certain time. The second button clears the field displaying the longitude and latitude. The third button stops the automatic location update. Finally, the exit button returns to the login screen.

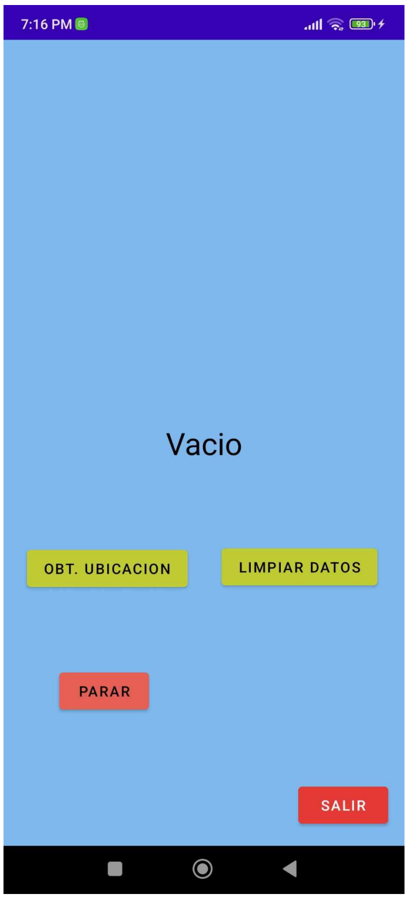


Figure 3. Get Location Screen.

There is also a screen to create a user (Figure 4), where certain fields such as ID, first name, last name, password, and role are requested. It is important to mention that the rate-limiting layer could be reflected on the screen to obtain the location. Since this screen updates the location periodically, exceeding the established number of requests in a given time will not display the corresponding error. Finally, the audit log layer is not reflected in the application as it is more of a backend layer that logs successful and unsuccessful requests made by the user.

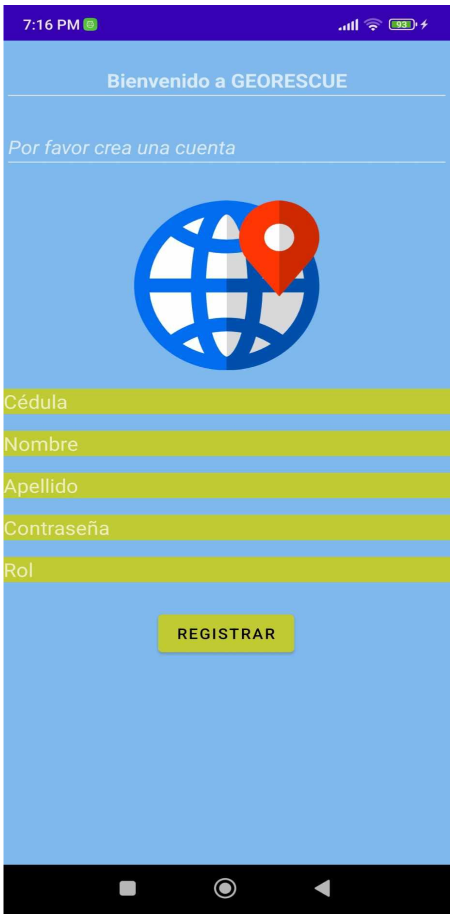


Figure 4. Create User Screen.